情報保護學會論文誌
第12券 第5號, 2002. 10

개선된 몽고메리 알고리즘을 이용한 저면적용 RSA 암호 회로 설계

김 무 섭*, 최 용 제*, 김 호 원*, 정 교 일*

Design of RSA cryptographic circuit for small chip area using refined Montgomery algorithm

Moo-seop Kim*, Yong-je choi*, Ho-won Kim*, Kyo-il Chung*

요 약

본 논문에서는 공개키 암호 시스템에서 인증, 키 교환 및 전자 서명을 위해 사용되는 RSA 공개키 암호 알고리즘의 효율적인 하드웨어 구현 방법에 대해 기술하였다. RSA 공개키 알고리즘은 모듈러 멱승 연산에 의해 계산되어지며, 모듈러 멱승 연산은 반복적인 모듈러 곱셈 연산을 필요로 한다. 모듈러 곱셈 구현을 위한 많은 알고리즘 중, 하드웨어 구현의 효율성 때문에 Montgomery 알고리즘이 많이 사용되어지고 있다. 지금까지 몽고메리 알고리즘을 이용하여 고성능의 RSA 암호회로를 설계하는 연구는 많이 수행되어 왔으나, 대부분의 연구가 시스템의 고성능을 위한 연산 시간의 감소에 중점을 두고있다. 하드웨어 구현에 제한이 있는 시스템에서 하드웨어 설계 시 가장 고려해야 할 사항은 시스템의 성능과 면적을 고려한 설계이다. 이러한 이유로, 본 논문에서는 기존의 Montgomery 알고리즘을 저면적 회로에 적합한 구조로 개선하였으며, 개선된 알고리즘을 이용하여 ETRI에서 개발한 스마트 카드용 에뮬레이팅 시스템인 IESA 시스템에 적용하여 검증하였다.

ABSTRACT

This paper describes an efficient method to implement a hardware circuit of RSA public key cryptographic algorithm, which is important to public-key cryptographic system for an authentication, a key exchange and a digital signature. The RSA algorithm needs a modular exponential for its cryptographic operation, and the modular exponential operation is consists of repeated modular multiplication. In a numerous algorithm to compute a modular multiplication, the Montgomery algorithm is one of the most widely used algorithms for its conspicuous efficiency on hardware implementation. Over the past a few decades a considerable number of studies have been conducted on the efficient hardware design of modular multiplication for RSA cryptographic system. But many of those studies focused on the decrease of operating time for its higher performance. The most important thing to design a hardware circuit, which has a limit on a circuit area, is a trade off between a small circuit area and a feasible operating time. For these reasons, we modified the Montgomery algorithm for its efficient hardware structure for a system having a limit in its circuit area and implemented the refined algorithm in the IESA system developed for ETRI's smart card emulating system.

Keyword : *RSA, Montgomery algorithm, Modular multiplication, Modular exponential*

* 한국전자통신연구원 정보보호기술연구본부{(gomskim, choiyj, khw)@etri.re.kr}

Ⅰ. 서 론

오늘날 정보화 사회에서 음성, 화상, 데이터 등과 같이 다양한 종류의 정보를 교환하고 저장하는 시스템에서 데이터 및 시스템의 신뢰성과 안전성은 필수적이며, 이러한 신뢰성을 제공하기 위해 다양한 암호 알고리즘이 적용되어 사용되어지고 있다. 공개키 암호 시스템은 다양한 공개키 기반 시스템에서 인증, 키 교환 및 전자서명과 같은 보안 특성을 제공한다. 공개키 암호 알고리즘 중 RSA 암호 알고리즘은 암호화 연산 또는 전자서명을 위해 가장 많이 쓰이는 암호 알고리즘이며, SET(Secure Electronic Transaction) 프로토콜과 같이 카드를 이용한 전자 상거래 및 각종 출입 통제 시스템 등에 채택되어 사용되어지고 있다.

RSA 암호 알고리즘은 두 개의 큰 소수를 곱하기는 쉽지만, 역으로 큰 수를 소인수 분해하기는 어렵다는 사실을 그 안정성의 근간으로 하고 있다. 일반적으로 이러한 RSA 암호 시스템은 시스템의 안정성을 높이기 위해 주로 1024 비트 이상의 큰 수를 기반으로 한 모듈러 멱승 연산을 수행한다. 그러나 1024 비트 이상의 큰 수에 대한 모듈러 연산은 소프트웨어로 구현하기에 많은 연산 시간을 필요로 하므로 대부분 고속 성능을 갖는 전용 프로세서 형태로 구현되고 있다.

RSA 암호 연산을 위한 전용 프로세서는 일반적으로 데이터의 저장을 위한 레지스터, 쉬프트 레지스터 및 덧셈기와 같은 연산장치 등으로 구성되어진다. 이러한 구성 요소들은 1024 비트 이상의 큰 데이터를 처리하도록 설계되므로 일반적으로 구현에 큰 면적을 필요로 한다. 지난 수년 동안 RSA 암호 회로의 효율적인 구현을 위한 많은 방법이 연구되어 왔으며, 이들 방법 중 대부분은 시스템의 연산 속도를 증가시키는 연구에 초점이 맞춰졌다. 이러한 RSA 암호 회로들은 충분한 성능을 제공하는 반면, RSA 암호 회로를 사용하는 시스템에서 충분한 면적을 제공할 수 없는 경우에는 적용할 수 없다는 단점이 있다.

이러한 제한적 시스템들은 일반적으로 암호 회로뿐 아니라 마이크로프로세서, MMU(Memory Management Unit), 메모리 장치 및 인터페이스 장치 등을 내장한 임베디드 시스템으로서 RSA 암호 회로를 적용하기 위한 회로의 면적에는 전용 프로세서와 달리 어느 정도의 제약이 존재한다.

본 논문은 암호 회로 구현에 필요한 면적에 이러한 제한이 있는 시스템에 적합한 RSA 암호 알고리즘의 효율적인 설계를 위해, 일반적으로 RSA 암호 시스템의 구현을 위해 많이 채택되고 있는 Montgomery 알고리즘을 저면적 회로에 적용 가능하게 개선하였으며, 개선된 Montgomery 알고리즘을 이용하여 회로 구현에 면적 제한이 많은 시스템에 적합한 구조를 설계하였다. 개선된 구조를 이용하여 설계된 RSA 암호 회로는 IC카드 개발용 에뮬레이터 시스템에 적용하여 동작 및 성능을 검증하였다.

본 논문은, 먼저 2장에서 RSA 암호 알고리즘의 개략적인 특성을 살펴보고, 3장에서는 RSA 암호 알고리즘의 수행에 필요한 Montgomery 알고리즘을 소개한 후, 문제점을 분석하여 32비트의 구성을 갖는 면적 제한 시스템에 적합한 구조를 갖도록 Montgomery 알고리즘을 개선한다. 개선된 알고리즘을 적용한 RSA 암호 회로의 구현을 4장에서 논의하고, 5장에서 설계된 회로의 분석을 끝으로 결론을 맺었다.

Ⅱ. RSA 암호 알고리즘

1976년 Diffie와 Hellman에 의해 공개키 암호 방식이 제안된 이후, RSA 암호 알고리즘은 공개키 암호 시스템에 가장 많이 사용되는 알고리즘 중 하나가 되었다. RSA 알고리즘은 암호화와 전자서명을 모두 지원하며, 큰 수의 경우 소인수분해가 어렵다는 사실에 그 안정성을 두고 있다.

RSA 암호 알고리즘은 안전성을 위해 보통 1024 비트 이상의 크기를 갖는 모듈러 값 M과 메시지 X를 사용하며, 키 생성 과정을 통해 생성한 공개키 e와 $X = X^{ed} \bmod M$인 관계를 갖는 개인키 d를 필요로 한다. 이러한 공개키 e와 개인키 d를 이용하여, RSA 암호 알고리즘은 암, 복호화 연산을 위해서 식 (1)과 같은 모듈러 멱승 연산을 수행한다.

$$C = X^e \pmod M$$
$$X = C^d \pmod M \quad (1)$$

식 (1)에서 볼 수 있듯이 RSA 암호 알고리즘은 공개키 e와 개인키 d에 대해 모듈러 멱승 연산을 수행하며, 모듈러 멱승 연산은 반복적인 모듈러 곱셈 연산에 의해 수행되어진다.

식 (1)의 RSA 암호 연산의 구현을 위해 설계된

회로는 일반적으로 암, 복호화 연산을 위해 사용되는 키의 길이가 증가할수록 bandwidth의 한계, 데이터의 처리 및 저장 공간의 증가, 소비 전력의 증가 등과 같은 문제점이 발생한다. 따라서 회로 구현을 위한 면적에 일정한 제한이 있는 시스템에 적합한 RSA 암호 회로를 설계하기 위해서는 이러한 문제점들을 적절히 고려한 설계가 필요하다.

Ⅲ. Montgomery Modular Multiplication

단일 z가 정수일 때, $z \bmod m$을 모듈러스 m에 대한 z의 모듈러 리덕션(modular reduction)이라 하며, $z \bmod m$은 z가 m에 의해 나누어진 후의 $[0, m-1]$에 존재하는 정수의 나머지를 의미한다.

모듈러 곱셈 연산을 구현하는 가장 기본적인 방법으로 식 (2)에서와 같이 임의의 다정도 정수 A와 B의 곱을 다정도 정수인 모듈러스 m에 대해 리덕션 연산을 추가적으로 사용하는 방법을 들 수 있다.

$$A \cdot B \bmod m = A \cdot B - \lfloor \frac{A \cdot B}{m} \rfloor \cdot m \qquad (2)$$

식 (2)의 리덕션 연산을 수행하기 위해서는 다정도 정수의 나눗셈 알고리즘을 사용하여야 하며, 이러한 모듈러 곱셈 연산은 보통 1024 비트 이상의 길이를 갖는 데이터를 연산하는 RSA 암호연산을 수행하기 위해 많은 시스템 자원을 필요로 하므로 회로의 구현에 적용하기에는 어려움이 있다.

3.1 Montgomery 곱셈 알고리즘

식 (2)에서와 같이 다정도 정수에 대한 나눗셈 연산은 연산에 많은 시간을 필요로 하므로 비효율적이며, n 비트의 다정도 정수 A와 B의 곱의 결과를 저장하기 위해 거의 $2n$ 비트에 가까운 레지스터 또는 메모리 장치를 필요로 하므로 하드웨어 부담 역시 증가하게 된다.

이러한 문제를 해결하는 보다 효율적인 방법으로 P. L. Montgomery[1]가 제안하는 모듈러 곱셈 알고리즘을 들 수 있다.

Montgomery 모듈러 곱셈은 모듈러 리덕션 연산의 효율적인 수행을 위해 데이터들을 정수 $R = b^n$ 을 이용하여 새로운 방법으로 나머지 클래스(residue class)로 표시하며, 새로 표시되는 클래스 내에서

알고리즘 1. Montgomery 모듈러 곱셈

input : $m=(m_{n-1}\cdots m_1 m_0)_b$, $x=(x_{n-1}\cdots x_1 x_0)_b$
$y=(y_{n-1}\cdots y_1 y_0)_b$, $x>0$, $m>y$, $R=b^n$,
$\gcd(m,b)=1$, $m'=-m^{-1} \bmod b$

1. $A=0$ *where* $A=(a_n\cdots a_1 a_0)_b$
2. for $i = 0$ *to* $n-1$
3. $\quad u_i=(a_0+x_i y_0)m' \bmod b$
4. $\quad A=(A+x_i y+u_i m)/b$
5. *end loop i*
6. if $A>m$ *then* $A=A-m$
7. *return* A

output : $xyR^{-1} \bmod m$

모듈러 연산을 새로 정의하여 모듈러 곱셈을 계산한다. 따라서 Montgomery 곱셈 알고리즘의 연산 결과는 원래 원하는 결과값 $xy \bmod m$이 아닌 다른 $xyR^{-1} \bmod m$의 형태로 표현되며, 원래의 결과값을 얻으려면 추가로 한 번의 Montgomery 곱셈 연산을 더 수행해야 한다.

Montgomery 모듈러 곱셈은 식 (2)에서와 같이 다정도 정수 x와 y의 모듈러 곱셈 연산을 두 개의 다정도 정수 x와 y의 곱을 한 번에 계산하지 않고, 알고리즘 1[4]에서 알 수 있듯이 $x_i y$의 값을 반복적으로 더하면서 리덕션 연산을 수행하면서 계산한다. 이러한 연산 단계를 수행함으로써 연산을 위한 반복 횟수는 증가하지만 계산과정에서 발생하는 결과값의 오버플로우 문제를 해결할 수 있으며, 모듈러 연산을 위해 복잡하고 시간이 많이 걸리는 나눗셈 연산을 사용하지 않고, 쉬프트 연산과 덧셈 연산만으로 모듈러 곱셈의 구현이 가능하므로 효율적인 하드웨어 구현을 가능하게 한다.

이러한 Montgomery 모듈러 곱셈의 효율적인 구현을 위한 많은 방법들이 제안되어 왔고, 현재 사용되고 있다. 가장 일반적인 구현 방법으로 CSA (Carry Save Adder) 구조를 사용한 모듈러 곱셈기[13]와 시스톨릭 어레이 구조를 사용한 모듈러 곱셈기[14]가 있으며, 구현에 많은 하드웨어를 필요로 하지만 고속연산이 가능한 radix 기반의 모듈러 곱셈기[15] 또는 RNS(Residue Number System) 기반의 모듈러 곱셈기[16]도 제안되고 있다. 그러나 이러한 방법들은 연산 성능은 만족하지만, RSA 암호 회로를 적용하는 시스템이 하드웨어의 면적에 제한적이면서 일정한 성능을 필요로 하는 경우에 적용하기에는 하드웨어의 면적이 너무 크다는 문제점이 있다.

따라서 제한적 시스템에 적합한 구조의 RSA 암호 회로를 위해서는, 이러한 문제점을 고려하여 적용하는 시스템의 특성에 적합한 연산단위로 알고리즘 1을 변형하여 설계할 필요가 있다. 본 논문에서는 일반적인 시스템과의 호환성을 고려하여 32비트의 연산 구조를 갖도록 설계하였다.

3.2 Montgomery 모듈러 곱셈 알고리즘의 수정

적용하려는 시스템의 목적에 적합한 RSA 암호 회로의 설계를 위해서는 먼저 알고리즘 1의 Montgomery 모듈러 곱셈 알고리즘의 분석을 통해 적용하려는 시스템의 목적에 맞는 적절한 형태로의 변환이 필요하다. 알고리즘 1의 Montgomery 모듈러 곱셈 알고리즘을 32비트 연산 구조를 사용한 저면적 회로에 적용하여 설계하려면 다음과 같은 몇 가지 문제를 가진다.

32비트 시스템은 내부에서 사용하는 데이터 버스의 폭이 32비트로 제한되어 있으므로, 보통 1024비트 이상의 큰 데이터를 사용하는 경우에는 32비트씩 데이터를 분할하여 전송, 처리하여야 한다. 이러한 시스템적 제약은 1024비트 이상 되는 데이터의 고속 처리를 필요로 하는 RSA 시스템용 모듈러 연산 장치의 구현을 어렵게 할 뿐 아니라 시스템의 성능도 저하시킨다.

이러한 문제점을 고려하여 Montgomery 모듈러 곱셈을 32비트 단위로 연산하는 경우, 알고리즘 1에서 3번 라인의 연산 $u_i=(a_0+x_i\cdot y_0)m' \bmod b$ 과 4번 라인의 $A=(A+x_iy+u_im)/b$ 연산 및 6번의 $A=A-m$ 연산은 1024비트 이상 되는 데이터 m, x, y들을 포함하고 있으므로, 32비트 연산에 적합하도록 다시 정의되어져야 한다.

$$u_i=(a_0+x_i\cdot y_0)m' \bmod b \tag{3}$$

먼저 식 (3)의 연산을 살펴보면, a_0와 y_0는 각각 1024비트 이상 되는 값들의 최하위 32비트의 워드 값을 나타내며, x_i는 반복되는 루프에서 데이터 x의 i번째에 해당하는 32비트의 워드 값을, m'값은 $m'=-m^{-1} \bmod b$ 값을 갖는 워드 값이다. 따라서 식(3)의 연산은 32비트 데이터들의 곱셈과 덧셈 연산으로 구성되므로 64비트의 연산 결과가 예측되지만, $\bmod b$ 연산을 고려하면 하위 32비트의 데이터 연산만 취함으로써 간단히 해결되어진다.

$$A=(A+x_iy+u_im)/b \tag{4}$$

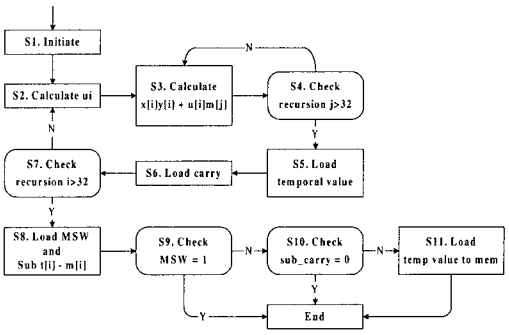
그러나 식(4)의 연산을 32비트 단위로 수행하기 위해서는 좀 더 많은 고려가 필요하다. 데이터 A, y, m 값들이 모두 1024비트 또는 그 이상의 값들을 가지므로 32비트 단위로 연산을 수행하기 위해서는 반복적인 연산이 필요하다.

이때 생각할 수 있는 간단한 방법이 식 (5)와 같이 2중 루프를 이용하여 32비트 연산을 반복적으로 수행하는 방법을 생각할 수 있다.

```
for i=0 to n-1
    for j=0 to n-1
        a_j = a_j + x_i y_j
    end loop j
    for j=0 to n-1
        a_j = a_j + u_i m_j
    end loop j
end loop i
```
(5)

식 (5)은 외부 루프 i에 대해, 첫 번째 내부 루프 j는 식 (4)에서 $A+x_iy$를 계산하고, 두 번째 내부 루프 j에서 이미 계산된 A에 u_im을 더해 줌으로써 $A+x_iy+u_im$을 두 번의 단계에 걸쳐 계산하고 있다.

그러나 두 번에 걸친 내부 루프 연산의 수행은 연산 시간의 급격한 증가를 초래한다. 예를 들어 1024비트 데이터의 연산을 수행하는 경우 32비트 단위로 연산을 수행하므로, i루프와 각 j루프를 모두 32번씩 반복해야하므로 $32\times(2\times32)$번의 루프를 반복해야 한다. 따라서 내부 루프의 연산 횟수를 줄일 수 있는 방안이 필요하다. 식 (4)의 구현을 위해 연산 횟수를 줄이기 위한 방법의 논리 흐름도를 [그림 1]에 나타내었다.



[그림 1] 모듈러 곱셈의 개선을 위한 논리 흐름도

〔그림 1〕의 논리 흐름도를 따라 32비트 연산 장치를 이용한 모듈러 곱셈의 개략적인 동작을 〔그림 2〕에 도시하였다.

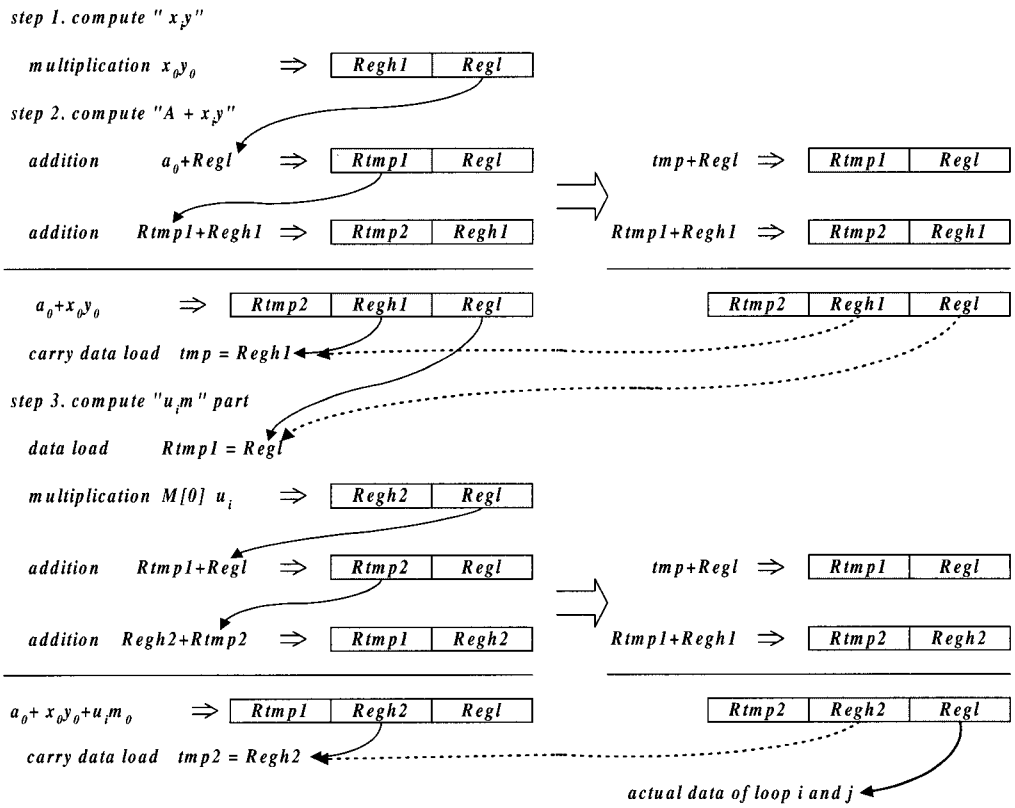
먼저, 임의의 i번째 루프에서 $x_i y$값을 계산하기 위해 y의 최하위 워드 y_0부터 시작하여 $x_i y_0$ 값을 계산하면, x_i와 y_0 모두 32비트의 값을 가지므로 $x_i y_0$ 곱셈은 32비트의 값을 갖는 *Regh*1 값과 *Regl* 값을 각각 얻을 수 있다. 이 결과 값을 이용하여 $A+x_i y$를 계산하기 위해 a_{ij}값을 곱셈 $x_i y$ 결과의 하위 32비트 값 *Regl*과 더해준다. 여기서 a_{ij} 값은 1024비트 이상의 A에서 식(5)에서의 루프 i에서 j번째에 해당하는 A값의 워드를 의미한다.

다시 이 연산 결과에 $u_i m_0$ 값을 더해주기 위해 동일한 방법으로 32비트 데이터들의 곱셈 연산을 수행하며, $A+x_i y$를 계산한 하위 32비트 데이터 값 *Regl*과 더해준다. 두 번의 곱셈 연산과 이와 관련된 덧셈 연산을 모두 수행하면 *Regl*은 $A+x_i y+u_i m$ 연산의 i번째 루프에서 y와 m의 최하위 워드값 y_0와 m_0에 의한 값 $A+x_i y_0+u_i m_0$의 하위 32비트 데이터 값을 저장하고 있다. 이제 $A+x_i y_0$를 계산한 상위 32비트의 값 *Regh*1과 $u_i m$을 더해준 경우 발생하는 32비트의 값 *Regh*2을 더해주면 연산의 i번째 루프에서 최하위 워드의 y_0와 m_0에 대해 연산을 수행한 상위 32비트의 값과 캐리 값을 구할 수 있다.

동일한 방법으로 다시 y_1과 m_1으로 인수를 증가하면서 연산을 수행하여 y와 m의 최상위 워드까지 연산을 반복하면 식 (4)의 값을 구할 수 있다.

그러나 2개의 32비트 값 *Regh*1, *Regh*2들의 덧셈을 위해서는 다시 한 번의 덧셈 연산을 추가로 필요로 한다. 이러한 추가적인 덧셈 연산은 외부 루프 i의 반복 횟수를 고려할 때, i번만큼의 덧셈 연산을 추가로 수행해야하므로 회로의 성능을 감소시킨다.

따라서 본 논문에서는 이러한 덧셈 문제를 해결하기 위해 〔그림 2〕에 도시한 것과 같이 2개의 상위 32비트 데이터 값 *Regh*1, *Regh*2들을 다음 반복 루프에서 연산을 수행할 때 입력 캐리 값으로 사용함으로써 추가적인 덧셈 연산에 소요되는 지연시간을 해소하였다. 이러한 경우 덧셈 연산이 더 증가하



〔그림 2〕 모듈러 곱셈기의 연산 블럭도

알고리즘 2. 수정된 Montgomery 곱셈

for $i = 0$ *to* $n-1$ *loop*
  $(Regh1, Regl) = x_i y_0 + a_0$
  $(R_{f1}, R_u) = m' Regl$
  $(Regh2, Regl) = m_0 R_u + Regl$
  for $j = 1$ *to* $n-1$ *loop*
    $(Regh1, Regl) = a_i y_j + a_j + Regh1$
    $(Regh2, Regl) = m_j R_u + Regl + Regh2$
    $a_{j-1} = Regl$
  end loop j
  $(R_p, Regl) = Regh1 + Regh2$
  $(R_{f1}, Regl) = a_n + Regl$
  $a_{n-1} = Regl$
  $(R_2, R_p) = a_{n+1} + R_p$
  $(R_{f1}, R_p) = R_{f1} + R_p$
  $a_n = R_p$
end loop i

게 되나 어레이 곱셈기와 데이터 선택기를 적절하게 사용하여 곱셈 연산과 덧셈 연산을 동시에 수행함으로써 해결할 수 있다.

이렇게 수정된 모듈러 곱셈 알고리즘을 정리하면 알고리즘 2와 같으며, 알고리즘 2는 알고리즘 1에서 가장 많은 연산 시간이 소요되는 2~5번 단계를 32비트 연산 시스템에서 구현하기 위해 32비트 곱셈기와 덧셈기를 이용하여 연산하는 과정을 정리한 것이다. 알고리즘 2에서 내부 루프 j의 반복 횟수가 외부 루프 i보다 적은 이유는, i와 j가 모두 0일 때는 캐리 데이터를 고려할 필요가 없기 때문이다.

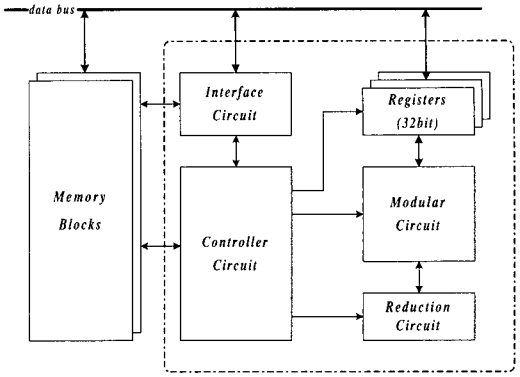
알고리즘 2에서 알 수 있듯이 수정된 Montgomery 모듈러 곱셈 알고리즘은 모듈러 곱셈 연산을 수행할 때 1024비트 이상의 큰 데이터를 사용하지 않고, 32비트 데이터와 연산장치만을 이용하여 연산을 수행함을 알 수 있으며, 식 (5)에서 두 번에 걸쳐 발생하였던 내부 연산 루프를 한 번만 수행하고 있음을 알 수 있다.

Ⅳ. Montgomery 모듈러 곱셈기 및 RSA 암호 회로의 하드웨어 구현

본 절에서는 32비트 시스템에 적합하게 변형한 알고리즘 2를 위한 Montgomery 모듈러 곱셈기를 설계하고, 설계된 모듈러 곱셈기를 이용한 RSA 암호 회로의 설계에 대해 기술한다.

4.1 아키텍쳐 개요

하드웨어 구현에 제한이 많은 시스템에 효율적인 RSA 암호 회로를 위해 본 논문에서 설계한 RSA



〔그림 3〕 RSA 암호 프로세서의 구조

암호 회로의 구조는 〔그림 3〕과 같으며, 설계에 다음과 같은 구조적 특징을 고려하였다.

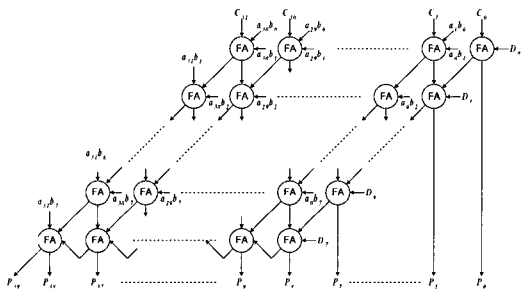
첫째, RSA 암호 회로 내부의 모든 데이터는 32비트 데이터 단위로 전송 및 연산 되어야한다. 설계 회로의 빠른 연산을 위해 데이터 경로를 증가시키면 연산 과정에서 발생하는 중간값을 저장하는 레지스터의 면적 역시 증가하게 된다. 따라서 내부의 연산뿐만 아니라 입출력을 위한 데이터 레지스터 역시 32비트로 구성한다.

둘째, 설계되는 RSA 암호 회로는 데이터 처리에 필요한 데이터를 저장하기 위해 별도의 레지스터를 사용하지 않고 시스템 내부의 메모리를 직접 access 할 수 있어야한다. 알고리즘 1의 경우 모듈러 곱셈 연산을 수행하기 위해서는 1024비트 이상 되는 데이터 m, x, y 값뿐만 아니라 연산의 중간과정에서 발생하는 데이터를 저장하기 위해 역시 1024비트 이상 되는 큰 레지스터가 필요하다. 그러나 이러한 레지스터들은 구현에 많은 칩 면적을 필요로 한다. 따라서 적은 면적에서도 효율적인 연산의 수행을 위해, 별도의 데이터 저장용 레지스터를 사용하지 않고 내부 메모리 장치만을 사용하여 연산을 수행할 수 있게 설계한다.

셋째, RSA 암호 회로를 적용하는 시스템에서 사용하기 편리하게 외부에서 인가되는 제어 신호에 따라 모듈러 곱셈 연산과 모듈러 멱승 연산을 선택적으로 연산할 수 있도록 설계한다.

4.2 모듈러 곱셈기의 구현

알고리즘 2의 Montgomery 모듈러 곱셈 알고리즘을 구현하기 위해서는 32비트의 데이터들을 곱셈 연산하기 위한 곱셈기가 필요하다. 그러나 일반적으로



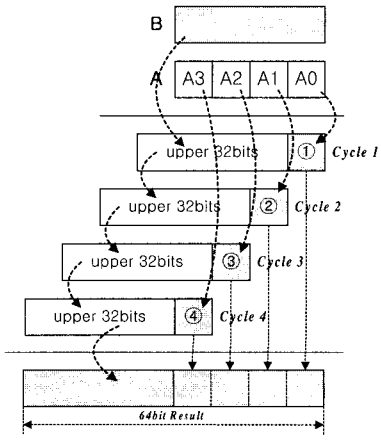
〔그림 4〕 32bit×8bit array 곱셈기

32비트 곱셈기는 구현에 필요한 회로 면적이 클 뿐 아니라 연산 속도도 느린 단점이 있다.

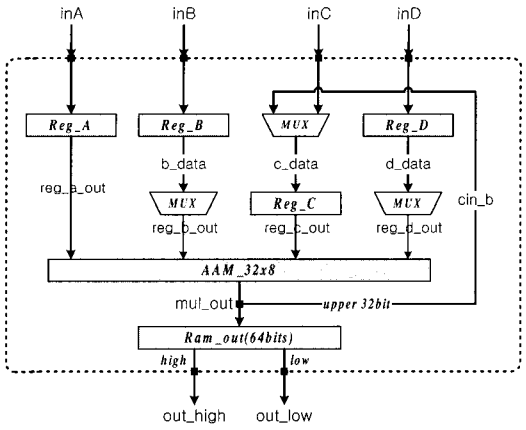
따라서 〔그림 4〕에 나타낸 32비트와 8비트 데이터를 곱하면서 더하는 array 곱셈기[5,6]를 기본 연산 장치로 하여 〔그림 5〕에 도시한 바와 같이 반복 연산을 통해서 32비트의 데이터를 곱하는 방법을 사용하였다.

〔그림 4〕의 array 곱셈기를 이용하여 〔그림 6〕의 32비트 곱셈기를 구현하기 위해서는, 〔그림 5〕와 같이 어레이 곱셈기를 4번에 걸쳐 반복하여 연산을 수행해야 하며, 각 반복 연산마다 상위 32비트 값은 다음 곱셈 단계에서 캐리로 인가되어 더해진다. 이러한 구조를 회로 모듈로 구현하면 〔그림 6〕과 같은 구조를 가지며, 〔그림6〕의 곱셈기는 64비트의 출력 데이터를 32비트씩 나누어서 사용하도록 하고 있다.

알고리즘 2의 각 단계별 연산을 적절히 수행하기 위해서는 〔그림 6〕의 곱셈기와 데이터 선택 장치, 그리고 32비트 덧셈기 회로를 추가적으로 사용하여 구현 할 수 있다.



〔그림 5〕 어레이 곱셈기 연산



〔그림 6〕 32bit×32bit 곱셈기

한 번의 곱셈 연산을 위해서 4번의 곱셈 연산을 반복 수행하는 것은 알고리즘 2에서 볼 수 있듯이 모듈러 곱셈 연산의 성능을 현저하게 감소시킴을 알 수 있다. 특히 반복 연산에 의한 성능 감소는 내부의 j루프 부분을 계산할 때 현저하게 나타난다.

이러한 구조적 문제를 어느 정도 보완하기 위해 보통 사용하는 방법으로 시스템 클럭을 증가시키는 방법이 있다. 그러나 회로의 latency 관점에서 볼 때, 시스템 클럭의 증가는 어느 정도 한계가 있다. 따라서 효율적인 모듈러 곱셈기의 구현을 위해서는 시스템 클럭의 증가와 함께 복잡한 연산의 순서를 체계적이면서 효율적으로 수행할 수 있는 제어 회로의 설계가 병행되어져야 한다.

4.3 RSA 암호 회로의 구현 및 검증

〔그림 1〕의 제어 흐름을 갖는 제어회로와 〔그림 6〕의 곱셈기 회로를 이용하여 구현된 모듈러 곱셈기 회로를 이용한 RSA 암호 회로를 구현하기 위해서는, 알고리즘 3과 같은 Montgomery 모듈러 멱승을 수행하여야 한다.

알고리즘 3[4]에서 알 수 있듯이 Montgomery 모듈러 멱승은 Montgomery 모듈러 곱셈을 사용하여 제곱 연산과 반복되는 루프 i에서 i번째 암호키의 값 e_i가 1인지 여부에 따라 추가적인 모듈러 곱셈을 수행할 것인지 여부를 판단한다.

일반적으로 제곱 연산이 곱셈 연산보다는 연산 속도가 빠르므로 시스템의 성능 향상을 위해 제곱 연산 회로와 곱셈 연산 회로를 구분하여 구현할 수 있으나, 구현 면적에 제한이 있는 시스템에서는 회로를 구현할 면적에 제한이 있으므로 곱셈 연산 회로

알고리즘 3. Montgomery 모듈러 멱승

input : $m=(m_{n-1}\cdots m_0)_r, R=r^l, e=(e_t\cdots e_0)_2$
with $e_t=1, m'=-m^{-1} \bmod r$,
and *message* x, $1\le x\le m$.

1. $xp=xR \bmod m$, $A=R \bmod m$
2. for i = t *downto* 0
3. $A=MontMul(A,A)$
4. if $e_i=1$ *then* $A=MontMul(A,xp)$
5. *else* $A=A$
6. *end loop i*
7. $A=MontMul(A,1)$
8. *return* A

output : $x^e \bmod m$

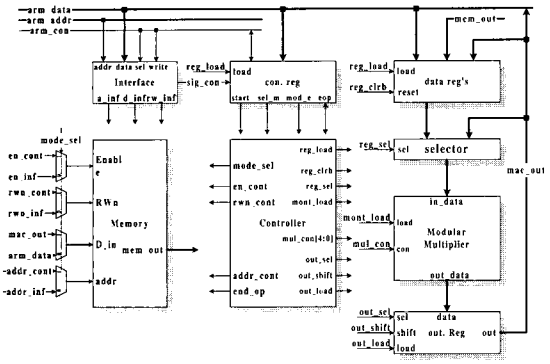
를 사용하여 제곱 연산과 곱셈 연산을 수행한다.

알고리즘 2의 구성을 갖는 Montgomery 모듈러 곱셈기를 이용하여 알고리즘 3의 Montgomery 모듈러 멱승 연산을 수행하는 RSA 암호 회로의 구성을 보면 〔그림 6〕과 같다.

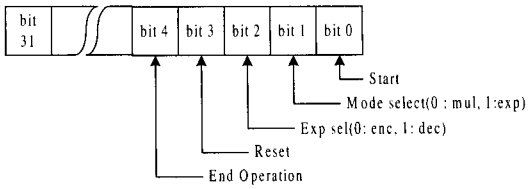
〔그림 7〕의 RSA 암호 회로의 동작을 개략적으로 살펴보면, 먼저, RSA 암호 회로의 연산을 수행하기 위해 상위 시스템에서는 연산에 필요한 데이터를 인터페이스 회로를 통해 메모리에 저장한 후 연산을 수행하라는 제어 신호를 RSA 암호회로에 인가한다.

RSA 암호 회로는 인터페이스를 통해서 인가된 제어 신호가 컨트롤 레지스터의 동작 시작 비트를 세팅하면, 설정된 값에 따라 해당되는 연산을 수행한다. 컨트롤 레지스터의 지시에 따른 연산을 수행한 후, RSA 암호 회로는 결과값을 메모리에 저장하고, 제어회로는 연산이 끝났음을 알리는 인터럽트 신호를 컨트롤 레지스터에 세팅한다.

상위 구동 시스템은 RSA 암호 회로가 인터럽트 신호를 발생하면 지시한 연산이 종료되었음을 알고, 인터페이스 회로를 통해 메모리의 결과값을 읽어들



〔그림 7〕 RSA 암호 회로의 블록 구성도



〔그림 8〕 컨트롤 레지스터의 비트별 설정 값

여 필요한 연산을 수행한다.

〔그림 7〕에서 인터페이스 회로는 RSA 암호 회로를 구동하는 상위 시스템과 RSA 암호 회로와의 데이터와 제어 신호를 전송하며, 컨트롤 레지스터는 〔그림 8〕에서 볼 수 있듯이 RSA 암호 회로의 초기화, 연산의 시작 및 수행할 연산을 설정하고, 연산이 종료하였음을 알려주는 인터럽트 신호를 발생시킨다.

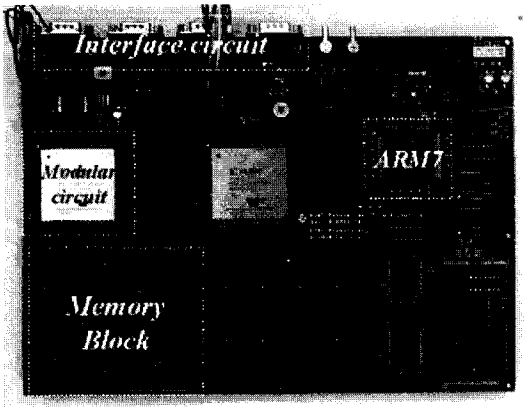
제어회로는 컨트롤 레지스터에 설정된 동작에 따라 RSA 암호 회로가 모듈러 곱셈 또는 모듈러 멱승 연산을 수행하는데 필요한 데이터 경로를 설정하며, 연산의 수행에 필요한 제어 신호를 발생한다.

설계된 RSA 암호 회로에서 〔그림 7〕의 데이터 레지스터들은 32비트 단위로 이루어지는 연산의 수행을 위해서 32비트의 길이를 가지며, 연산 과정에서 발생하는 데이터들을 선택적으로 저장하기 위하여 데이터 선택기들과 함께 사용되어진다.

알고리즘 2는 각 반복 루틴마다 필요한 값을 메모리에서 읽어와야 하므로 많은 메모리 엑세스 시간을 필요로 한다. 연산에 필요한 데이터를 읽고, 연산의 중간값을 메모리에 저장하는 반복작업이 많으므로 발생할 수 있는 동작 지연시간을 줄이고, 보다 효율적인 리덕션 연산을 위해 출력 레지스터를 추가로 사용하여 선택적으로 데이터를 저장하도록 설계하였다.

상기한 바와 같이 설계된 RSA 암호 회로는 구현 면적에 제한이 있는 시스템에의 적합성을 시험하기 위하여 시스템 구현에 제약적 요인이 많은 IC카드 시스템을 시험 대상으로 선정하였으며, 〔그림 9〕와 같이 ETRI에서 개발한 IC카드용 에뮬레이션 시스템인 IESA 시스템에 적용하여 동작을 검증하였다.

동작이 검증된 RSA 암호 회로는 0.5㎛ CMOS 셀 라이브러리와 Synopsis 툴을 이용하여 합성 한 결과, RSA 암호 코어는 27,000 게이트로 구현되었으며, 코어의 구동을 위해서는 4Mbits 미만의 메모리가 필요하였다. 설계된 RSA 암호 회로의 특징과 기존 시스템과의 성능 비교는 〔표 1〕과 같다.



〔그림 9〕 설계된 RSA 암호 회로 검증

〔표 1〕 설계된 RSA 암호 회로의 특징

	〔18〕	〔19〕	proposed
합성공정	0.35㎛	0.4㎛	0.5㎛
동작 주파수	60㎒	20㎒	40㎒
게이트 수	18K	40K	27K
사용 메모리	256×32와 64×32 RAM		128×32 RAM
모듈러 곱셈*			356㎲
암호화**			6.4㎳
복호화*	60㎳	650㎳	510㎳

* : 1024비트 키 또는 데이터 연산
** : 16비트 키 연산(2^{16})

〔표 1〕에서 볼 수 있듯이 본 논문에서의 방법은 실제로 IP 형태로 제작되어 판매되고 있는 시스템[18] 또는 다른 논문[19]과 비교할 경우 코어 게이트 수는 약간 더 많아 보이나 실제로 연산을 수행하는 경우 칩 내부에서 사용하는 메모리 양이 매우 적으므로, 실제 칩 면적에서 고려할 때 아주 작은 면적을 사용함을 알 수 있다. 그리고, 일반적으로 IC카드 시스템에서 초기 챌린지 신호가 인가된 후 기대되는 응답 속도가 1초 이내임을 고려할 때, 〔표 1〕에 나타낸 결과는 설계된 RSA 암호 회로가 구현 면적에 제한이 있는 시스템의 대표적인 예로 들 수 있는 IC 카드 시스템에서도 적절한 동작 특성을 가짐을 보여주고 있다.

V. 결 론

본 논문에서는 공개키 암호 시스템에서 인증, 키 교환 및 전자 서명을 위해 사용되는 RSA 공개키 암호 알고리즘을 면적 제한이 있는 시스템에 효율적으로 적용 할 수 있는 하드웨어 구현 방법에 대해 기술하였다. 일반적인 Montgomery 모듈러 곱셈 알고리즘을 하드웨어 구현에 면적 제한이 있는 시스템에 적합하도록 32비트 연산 단위를 이용한 형태로 변환하였으며, 변환된 Montgomery 모듈러 곱셈 알고리즘을 기반으로 하여 32비트 연산 장치를 이용한 RSA 공개키 암호 회로를 설계하였다.

본 논문에서 제안된 RSA 암호 회로는, 구현에 필요한 면적을 최소화하기 위하여 모든 연산을 32비트 단위로 처리하고 있으며, 내부에 많은 면적을 차지하는 레지스터들을 사용하지 않고, 시스템 메모리를 직접 엑세스하여 연산을 수행하도록 설계되었다. 설계된 RSA 암호 회로에서 연산에 가장 중요한 영향을 미치는 어레이 곱셈기의 성능은 설계 시 사용되는 공정에 의존적이며, 집적 회로로 구현되는 경우 최적화된 공정 라이브러리로 대체 되어질 수 있을 것이다. 본 논문에서는 0.5㎛ 공정을 사용하였으며, 코어 면적 27,000 게이트에서, 40㎒의 동작 주파수에서 1024비트의 모듈러 곱셈 연산에 356㎲, 16비트 길이의 암호 키를 이용한 1024비트 RSA 암호화 연산은 6.4㎳, 1024비트의 복호화 키를 이용한 연산은 510㎳ 이하의 연산 속도를 보였다. 이러한 동작 특성은 시스템의 구현에 면적 제한이 많은 시스템의 대표적인 예로 들 수 있는 IC카드 시스템에 적용 가능함을 알 수 있었다. 설계된 RSA 암호 회로 코어는 IP(Intellectual Property)형태로 가공하여 IC카드나 전자주민증 등과 같이 시스템의 면적 제한이 있는 소형 정보 단말기에 적용되어 질 수 있을 것으로 판단된다.

참 고 문 헌

〔1〕 P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, Vol. 44, No. 170, pp. 519~521, Apr. 1985.

〔2〕 S. R. Dusse and B. S. Kaliski, Jr., "A Cryptographic Library for the Motorola DSP56000," *Advances in Cryptology - Eurocrypt 90, LNCS, No. 473*, Springer-Verlag, pp. 230~244, 1990. .

〔3〕 D. E. Knuth, *The Art of the Computer*

Programming, Vol. 2, Addison-Wesley, 3rd ed., 1998.

[4] A. J. Menezes, P. C. van Oorschot, and S. C. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[5] Kai Hwang, *Computer Arithmetic : Principles, Architecture, and Design*, John Wiley & Sons, 1979.

[6] B. Parhami, *Computer Arithmetic*, Oxford University Press, 2000.

[7] S. E. Eldridge and C. D. Walter, "Hardware Implementation of Montgomery's Modular Multiplication Algorithm," *IEEE Transaction on Computers*, Vol. 42, No. 6, 1993.

[8] F. Tenca and C. K. Koc, "A scalable architecture for Montgomery multiplication," *CHES 99, LNCS No. 1717*, Springer-Verlag, pp. 94~108, 1999.

[9] T. Blum and C. Parr, "Montgomery modular exponentiation on reconfigurable hardware," *14th IEEE Symposium on Computer Arithmetic*, pp. 70~77, 1999.

[10] S. E. Eldridge, "A faster modular multiplication algorithm," *Int. J. Computer Math.*, Vol. 40, pp. 63~68, 1991.

[11] M. Shand and J. Vuillemin, "Fast Implementation of RSA cryptography," *11th IEEE Symp., on Computer Arithmetic*, pp. 252~259, 1993.

[12] D. R. Stinson, *Cryptography, Theory and Practice*, CRC Press, 1995.

[13] T. W. Kwon, J. R. Choi and etc., "Two implementation methods of a 1024-bit RSA cryptoprocessor based on modified Montgomery algorithm," *Circuits and Systems, ISCAS 2001*, Vol. 4, pp. 650~653.

[14] C. K. Koc and C. Y. Hung, "Bit-level systolic arrays for modular multiplication," *Journal of VLSI Signal Processing*, pp. 215~223, 1991.

[15] N. Takagi, "A radix-4 modular multiplication hardware algorithm for modular exponentiation," *IEEE Trans. Computers*, Vol. 41, pp. 949~956, 1992.

[16] A. A. Hiasat, "New efficient structure for a modular multiplier for RNS," *computers, IEEE Trans*, Vol. 49, pp. 170~174, 2000.

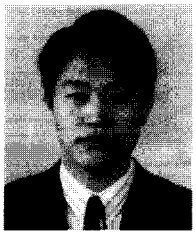
[17] 하재철, 문상재, "Montgomery 곱셈기를 이용한 효율적인 모듈라 멱승기 구조," *정보보호학회 논문지*, Vol. 11, No. 5, Oct., 2001

[18] www.sidsa.com/datasheets/RSA/ds_rsa2048a_short.html

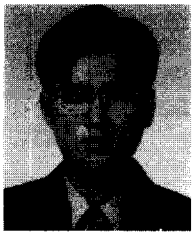
[19] Deng Yuliang, Mao Zhigang and etc., "Implementation of RSA cryptoprocessor based on Montgomery algorithm," *Solid-State and Int. Cir. Technology*, pp. 524~526, 1998.

[20] 서광석 외, *암호학과 대수학*, 북스힐, 1999.

〈著者紹介〉



김 무 섭 (Moo-Seop Kim)
1995년 2월 : 금오공과대학교 전자공학과(학사)
1998년 2월 : 경북대학교 전자공학과(석사)
1998년 2월~1999.8월 : LG종합기술원 Innovation Center 연구원
1999년 9월~현재 : 한국전자통신연구원 정보보호연구본부 연구원
〈관심분야〉 정보보호 및 응용, 암호회로 설계



최 용 제 (Yong-Je Choi)
1997년 2월 : 전남대학교 전자공학과(학사)
1999년 2월 : 전남대학교 전자공학과(석사)
1999년 9월~현재 : 한국전자통신연구원 정보보호연구본부 연구원
〈관심분야〉 암호모듈 설계, 정보보호

김 호 원 (Ho-Won Kim)
1993년 2월 : 경북대학교 전자공학과(학사)
1995년 2월 : 포항공과대학교 전자전기공학과(석사)
1999년 2월 : 포항공과대학교 전자전기공학과(박사)
1999년~현재 : 한국전자통신연구원 정보보호연구본부 선임연구원
〈관심분야〉 정보보호, 암호프로세서 설계, 타원곡선 암호모듈 설계

정 교 일 (Kyo-il Chung)
1981년 2월 : 한양대학교 전자공학과(학사)
1993년 8월 : 한양대학교 산업대학원 전자계산학과(석사)
1997년 8월 : 한양대학교 전자공학과(박사)
1981년 12월~현재 : 한국전자통신연구원 정보보호기술연구본부 부장/책임연구원
〈관심분야〉 IC Card, Security, Biometry, 정보전, 신호처리